IOT-BASED SMART PARKING SYSTEM

PROJECT CODE:

Raspberry Pi Script:

```
import RPi.GPIO as GPIO
import time
import serial
from RPLCD import CharLCD

# Define GPIO pins
IR_SENSOR = 18
SERVO = 12
GREEN_LED = 17
RED_LED = 27

# Define WiFi module serial port
WIFI_SERIAL_PORT = "/dev/ttyUSB0"

# Initialize LCD
lcd = CharLCD(numbering_mode=GPIO.BCM, cols=16, rows=2, pin_rs=26, pin_e=19, pins_data=[13, 6, 5, 11])

# Initialize WiFi module
wifi_serial = serial.Serial(WIFI_SERIAL_PORT, baudrate=9600, timeout=1)

# Set up GPIO mode and pins
GPIO.setmode(GPIO.BCM)
GPIO.setup(IR_SENSOR, GPIO.IN)
GPIO.setup(SERVO, GPIO.OUT)
GPIO.setup(GREEN_LED, GPIO.OUT)
GPIO.setup(RED_LED, GPIO.OUT)
```

```
def is_parking_spot_vacant():
    return GPIO.input(IR_SENSOR) == GPIO.HIGH

def move_servo(angle):
    pwm = GPIO.PWM(SERVO, 50)
    pwm.start(0)
    duty_cycle = 2 + (angle / 18)
    pwm.ChangeDutyCycle(duty_cycle)
    time.sleep(0.5)
    pwm.stop()

def update_leds(parking_status):
    if parking_status == "vacant":
        GPIO.output(GREEN_LED, GPIO.HIGH)
        GPIO.output(RED_LED, GPIO.LOW)
    elif parking_status == "occupied":
        GPIO.output(GREEN_LED, GPIO.LOW)
        GPIO.output(RED_LED, GPIO.HIGH)

def update_lcd(message):
    lcd.clear()
    lcd.write_string(message)

def send_wifi_command(command):
    wifi_serial.write((command + '\r\n').encode())
    response = wifi_serial.readline().decode()
    return response.strip()

try:
    while True:
```

```
        if is_parking_spot_vacant():
            update_leds("vacant")
            move_servo(90)
            update_lcd("Parking Vacant")
            send_wifi_command("PARKING_VACANT")
        else:
            update_leds("occupied")
            move_servo(0)
            update_lcd("Parking Occupied")
            send_wifi_command("PARKING_OCCUPIED")

        time.sleep(2)

except KeyboardInterrupt:
    GPIO.cleanup()
    wifi_serial.close()
```

Flutter-based parking system app:

```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ParkingAvailabilityScreen(),
    );
```

```
  }
}

class ParkingAvailabilityScreen extends StatefulWidget {
  @override
  _ParkingAvailabilityScreenState createState() =>
      _ParkingAvailabilityScreenState();
}

class _ParkingAvailabilityScreenState extends State<ParkingAvailabilityScreen> {
  // Add logic to receive and display parking availability data here

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Parking Availability'),
      ),
      body: Center(
        child: // Add widgets to display parking availability data here
      ),
    );
  }
}
```

HOW TO REPLICATE THE PROJECT, DEPLOY IOT SENSORS, DEVELOP THE TRANSIT INFORMATION PLATFORM, AND INTEGRATE THEM USING PYTHON:

Components Needed:

- Raspberry Pi (with power supply, microSD card, and keyboard/mouse/monitor for initial setup)
- IR sensor module
- Servo motor
- Wi-Fi module (e.g., ESP8266)
- LCD display (with compatible driver board)
- Jumper wires
- Breadboard or PCB
- Power supply for the Raspberry Pi and servo motor
- Mobile device (for testing the mobile application)

Step 1: Setting Up the Raspberry Pi

i. Install the Raspberry Pi OS on the microSD card. You can use the official Raspberry Pi Imager for this.
ii. Connect the Raspberry Pi to a monitor, keyboard, and mouse. Power it up and complete the initial setup.
iii. Connect the Wi-Fi module to the Raspberry Pi via the GPIO pins or USB, and configure it to connect to your local Wi-Fi network.

Step 2: Wiring the IR Sensor and Servo Motor

i. Connect the IR sensor to the Raspberry Pi's GPIO pins. Follow the datasheet or module documentation to find out which pins are used for power, ground, and signal.
ii. Connect the servo motor to the Raspberry Pi. You'll typically need to connect the power, ground, and signal wires to specific GPIO pins. Refer to the servo motor's datasheet or documentation for details.

Step 3: Programming the Raspberry Pi

i. Write Python scripts to interface with the IR sensor and servo motor. Use libraries like RPi.GPIO or pigpio to control the GPIO pins. These scripts will handle the detection of vehicles and the movement of the barrier.
ii. Test the IR sensor and servo motor individually to make sure they work correctly. For example, when a vehicle is detected, the barrier should open, and when no vehicle is detected, the barrier should remain closed.

Step 4: Setting Up the LCD Display

i. Connect the LCD display to the Raspberry Pi. Ensure you have the necessary libraries and drivers installed to interface with the display. This might involve some soldering or connecting via a compatible driver board.
ii. Write a Python script to display relevant information on the LCD display. This could include parking availability, status messages, etc.

Step 5: Developing the Transit Information Platform

i. Create a backend server for the transit information platform. You can use a framework like Flask or Django for Python.
ii. Set up a database to store information about parking spots, their availability, and any other relevant data.
iii. Develop APIs to interact with the database and serve information to the mobile application.

Step 6: Building the Mobile Application

i. Choose a platform for mobile app development (e.g., Android - Java/Kotlin, iOS - Swift/Objective-C, or cross-platform - Flutter or React Native).
ii. Create a user interface for the mobile application where users can view parking availability, reserve spots, and receive notifications.

iii. Implement the logic to communicate with the backend server using HTTP requests to fetch parking information and send reservation requests.

Step 7: Integrating the Components

i. Use the Wi-Fi module to allow the Raspberry Pi to connect to the internet. This will enable communication between the Raspberry Pi and the transit information platform.
ii. Integrate the Python scripts for the IR sensor, servo motor, and LCD display with the transit information platform. This could involve making HTTP requests to the server to get parking availability information and updating the display accordingly.
iii. Implement the logic for the mobile application to interact with the backend server for fetching parking information and sending reservation requests.

Step 8: Testing and Deployment

i. Test the entire system to ensure that the IR sensor, servo motor, LCD display, and mobile application work together seamlessly.
ii. Deploy the backend server and mobile application to a hosting platform of your choice. You may need a domain and SSL certificate for secure communication.
iii. Deploy the Raspberry Pi and associated hardware in the parking area where you want to implement the smart parking system.

Step 9: User Testing and Feedback

i. Invite users to test the system and gather feedback for improvements. Make any necessary adjustments based on user experience and performance.

EXAMPLE OUTPUTS OF RASPBERRY PI DATA TRANSMISSION AND MOBILE APP UI:

Vehicle Detected (via IR sensor):

JSON:


```
{
   "status": "vehicle_detected",
   "parking_spot_id": 1,
   "timestamp": "2023-10-30T14:30:00"
}
```


Vehicle Departed (via IR sensor):

JSON:


```
{
   "status": "vehicle_departed",
   "parking_spot_id": 1,
   "timestamp": "2023-10-30T15:00:00"
}
```


LCD Display Output:

Smart Parking System Available Spots: 10 Status: CLOSED

Mobile App User Interface:

Home Screen:

- Map view showing the parking area with marked spots (green for available, red for occupied).
- Total available spots count: 10

Parking Details Screen (when a spot is selected):

- Spot ID: 1
- Availability: Available/Occupied
- Distance from the user's current location: 50 meters
- Button to Reserve (if available)

Reservation Confirmation Screen:

- Spot ID: 1
- Estimated time to reach the spot: 5 minutes
- Countdown timer for reservation expiration: 10 minutes
- Button to Confirm Reservation

Navigation Screen:

- Map view with a route from the user's current location to the reserved spot.
- Estimated time of arrival (ETA): 5 minutes

Notification (when the reservation is about to expire):

- "Your parking reservation for Spot 1 is about to expire. Please confirm or vacate the spot."

Reservation Expired Screen:

- Message: "Your reservation for Spot 1 has expired. Please find another spot."

Feedback Screen (after parking):

- Rating system for the parking experience (1 to 5 stars).
- Optional comment box for feedback.